CV Homework 1

0856005 劉安齊
0856123 柯若霖
0853426 陳紹雲

Introduction

Camera calibration is used to estimate intrinsic and extrinsic parameters. Intrinsic parameters deal with the camera's internal characteristics, such as, focal length, skew, distortion, and image center. Extrinsic parameters describe object's position and orientation in the real world. Camera calibration is a necessary step in 3D computer vision in order to extract metric information from 2D images.

Camera calibration can also solve the distortion caused by camera.It is known that all the images captured from a camera should pass through an optical lens to map the 3D real world view to a CCD sensor. Because of the nature lens distortion, the shape of an original image is nonlinearly changed.

This assignment we used Zhang's method [1] to approach. Zhang's described here, use multiple views of a 3D pattern of known structure but unknown position and orientation in space.This methods exist that make no assumptions about the 3D structure of the scene, using multiple views of arbitrary, rigid structures. Based on the model described in Zhang's method, the following procedure are recovered:

1) Print a pattern and attach it to a planar surface.
2) Take a few images of the model plane under different orientations by moving either the plane or the camera.
3) Detect the feature points in the images.
4) Estimate the five intrinsic parameters and all the extrinsic parameters using the closed-form solution as described below.
5) Estimate the coefficients of the radial distortion by solving the linear least-squares.
6) Refine all parameters by minimizing.

Implementation procedure

Firstly, we used world coordinates and image coordinates to find Homography matrix which is the combination of intrinsic and extrinsic matrix. The equation is given at below:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underset{\text{up to a scale}}{\underline{\rho}} H \begin{bmatrix} U \\ V \\ 1 \end{bmatrix} \qquad \text{(eq. 1)}$$

where u, v are image coordinates and U, V are world coordinates. Since points on chessboard lie on the same plane, we don't need to consider the third dimension of world coordinate W.

We rewrote eq. 1 as a system of $Lx = 0$ shown below:

$$\begin{bmatrix} \widetilde{\mathtt{M}}^T & \mathbf{0}^T & -u\widetilde{\mathtt{M}}^T \\ \mathbf{0}^T & \widetilde{\mathtt{M}}^T & -v\widetilde{\mathtt{M}}^T \end{bmatrix} \mathbf{x} = \mathbf{0} \, .$$

where $\overline{M}$ is an object coordinate, u, v are values in image coordinate and x is Homography vector $[h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33}]^T$. To find nullspace of L, we performed SVD decomposition $L = U\Sigma V^T$ and found its nullspace at last row of V^T. After that, we solved ρ using $\rho = 1 / (\overline{h_3}\, objpoint)$, where $\overline{h_3}$ is third row vector of H. And found the mean value of ρ using all object points in same chessboard.

Once we found Homography matrix, we referred to the equations at slide p78 which are:

$$\mathbf{h}_1^\top \mathbf{K}^{-\top} \mathbf{K}^{-1} \mathbf{h}_2 = 0$$
$$\mathbf{h}_1^\top \mathbf{K}^{-\top} \mathbf{K}^{-1} \mathbf{h}_1 = \mathbf{h}_2^\top \mathbf{K}^{-\top} \mathbf{K}^{-1} \mathbf{h}_2 \qquad \text{(eq. 2)}$$

where h denotes column vector of H. Let $B := K^{-T}K^{-1}$. Since B is a positive definite matrix, it can be formed as:

$$\mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & b_{23} \\ b_{13} & b_{23} & b_{33} \end{pmatrix}$$

We rewritten eq. 2 as $Vb = 0$:

$$\begin{bmatrix} \mathbf{v}_{12}^T \\ (\mathbf{v}_{11} - \mathbf{v}_{22})^T \end{bmatrix} \mathbf{b} = \mathbf{0} \, .$$

where v_{ij} denotes:

$v_{ij} = [h_{i1}h_{j1},\ h_{i1}h_{j2} + h_{i2}h_{j1},\ h_{i3}h_{j1} + h_{i1}h_{j3},\ h_{i2}h_{j2},\ h_{i3}h_{j2} + h_{i2}h_{j3},\ h_{i3}h_{j3}]^T$

and b denotes:

$b = [B_{11}, B_{12}, B_{13}, B_{22}, B_{23}, B_{33}]^T$

We performed SVD decomposition to find b whch is nullspace of V. B is up to a factor of $K^{-T}K^{-1}$ which is $B = \lambda K^{-T}K^{-1}$. In order to solve λ, we assume K

$$= \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$
.

λ can be solved by following equations:

$$v_0 = (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2)$$
$$\lambda = B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]/B_{11}$$

Now K can be easily computed using Cholesky factorization. After solved intrinsic matrix K, since $H = K[r_1\ r_2\ t]$, the extrinsic parameters can be solved by:

$\lambda = 1 / \left\|K^{-1}h_1\right\|$

$r_1 = \lambda K^{-1}h_1$

$r_2 = \lambda K^{-1}h_2$

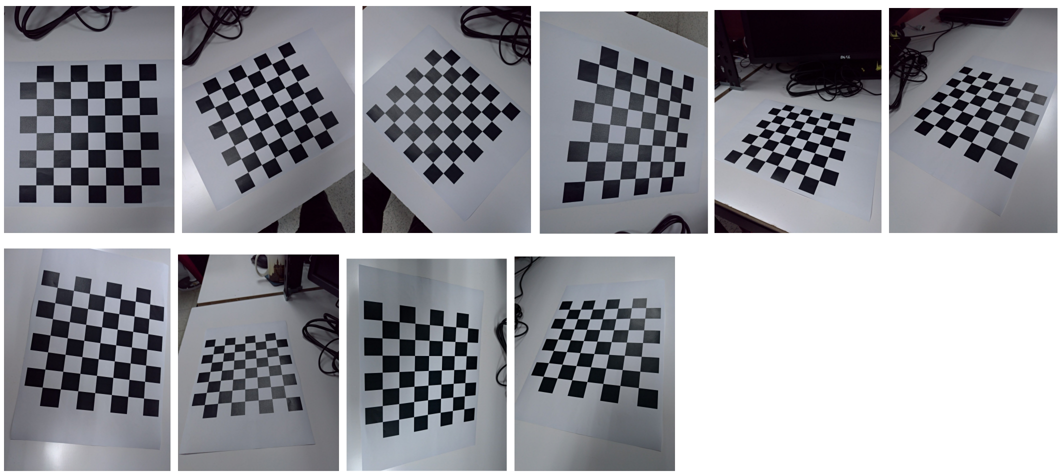
$r_3 = r_1 \times r_2$

$t = \lambda K^{-1}h_3$

Experimental Result

1. Sample Data

Photos:



Sample Data:

OpenCV:

intrinsic

```
[[3.17677580e+03 0.00000000e+00 1.64148698e+03]
 [0.00000000e+00 3.19707029e+03 1.43116619e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

extrinsics

```
[[-0.23877628 -0.14541111  0.13106983 -2.3977709  -2.52588459 11.15790517]
 [-0.61472654  0.66114584  1.36583168  0.79608187  0.63181685 16.74761483]
 [-0.83426321  0.26308385  0.44013934 -1.3732219  -1.35722488 16.45569686]
 [-0.92611891 -0.59340622 -1.0876699  -4.85358298  4.69578493 12.96550901]
 [-0.09744015  0.12689479  1.55340019  2.31166413 -0.46364501 10.72898312]
 [-0.26292545  0.0923835   1.06972383  0.39050736 -2.24259045 13.8781941 ]
 [-0.53226471  0.0521525   0.14657773 -1.11760407 -3.78366414 12.46140329]
 [-0.26652299  0.08769634  0.76062724 -1.10472584 -2.36466946 14.29506156]
 [-0.30852075  0.71608857  1.61400207  2.97994917 -1.87415744 12.63610036]
 [-0.46301626 -0.04234608  0.02577967 -1.42114755 -2.76418032
12.52289948]]
```

Our Work:

intrinsic

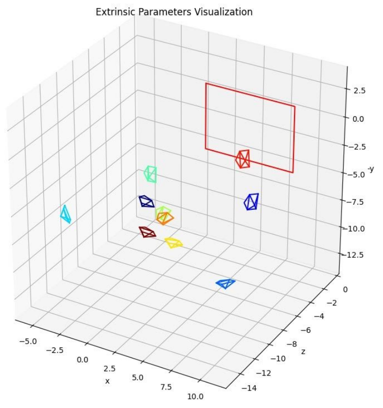
```
[[4.08724169e+03 9.19478294e+01 1.65302421e+03]
 [0.00000000e+00 3.90800537e+03 6.83366133e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

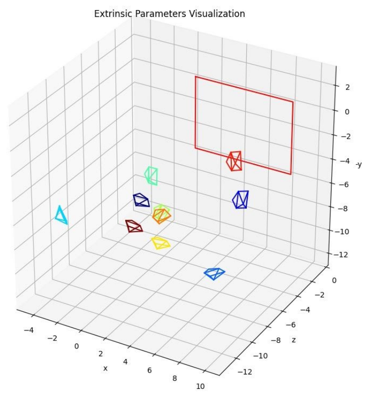
extrinsics

```
[[-1.74386999e-01 -1.75790503e-01  1.59074654e-01 -2.44366157e+00
   4.44575885e-02  1.40919059e+01]
 [-6.82684081e-01  8.62588631e-01  1.30315245e+00  6.06394456e-01
   4.66960367e+00  2.09819803e+01]
 [-9.65108672e-01  4.11237099e-01  4.06070602e-01 -1.45727154e+00
   2.50198183e+00  2.01741873e+01]
 [-1.19283122e+00 -7.18306641e-01 -1.00932213e+00 -4.76852154e+00
   7.71490450e+00  1.64026922e+01]
 [-2.46056605e-02  8.17207092e-02  1.56510610e+00  2.18540815e+00
   2.11223588e+00  1.35282710e+01]
 [-2.06690401e-01  6.76351286e-02  1.08959341e+00  3.25737627e-01
   1.03116523e+00  1.77352848e+01]
 [-6.99363413e-01  4.32009110e-03  1.55190747e-01 -1.09903834e+00
  -9.73194170e-01  1.60483278e+01]
 [-2.00286222e-01  9.24396612e-02  7.74643713e-01 -1.18882453e+00
   1.00307926e+00  1.83134090e+01]
 [-2.34370381e-01  9.19378707e-01  1.53865816e+00  3.03733104e+00
   1.11760694e+00  1.65369070e+01]
 [-5.71784066e-01 -1.33178029e-01  4.73270812e-02 -1.41644315e+00
   1.40140634e-01  1.57982756e+01]]
```

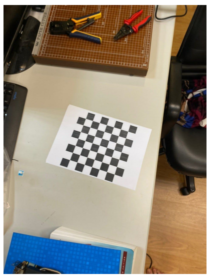
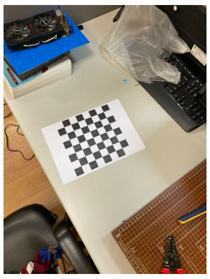
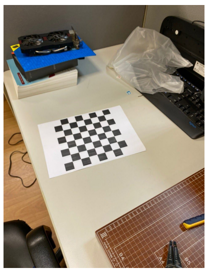
Visualization

opencv v.s. *our work*





2. Our Photos:

Data:

OpenCV:

intrinsic
```
[[971.91795802   0.         471.66412665]
 [  0.         980.43458945 622.16000833]
 [  0.           0.           1.        ]]
```

extrinsics
```
[[-3.07617685e-01  8.17647499e-02  3.13511705e-01 -2.42991399e+00
  -2.36675913e+00  2.83843063e+01]
 [-3.60953492e-01  1.75968052e-01  1.12911010e+00  6.01288301e-03
  -3.47577048e+00  2.99383392e+01]
 [-6.81918542e-01  1.01013395e-01  3.56207701e-01 -3.41484534e+00
  -3.52432339e+00  2.46271618e+01]
 [-6.01014308e-01  4.04028986e-01  1.19147057e+00  1.92830804e-01
  -2.55998536e+00  2.88348679e+01]
 [-6.77162170e-01  1.06719082e+00  1.87275654e+00  4.13887428e+00
  -2.62859592e-02  1.42568505e+01]
 [-7.95976523e-01  4.77454943e-01  1.15017378e+00  8.16910178e-01
  -2.60736063e+00  1.65417227e+01]
 [-6.68951912e-01  3.43845550e-01  1.22643754e+00  2.32445429e+00
  -3.62773625e+00  1.87244789e+01]]
```

Our work:

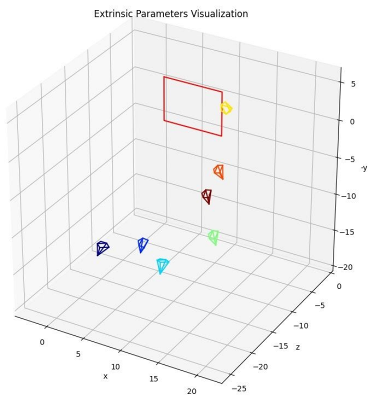
intrinsic
```
 [[ 1.57447314e+03  4.70020828e+01  5.81820358e+02]
 [ 0.00000000e+00  1.31871761e+03 -7.16085563e+01]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

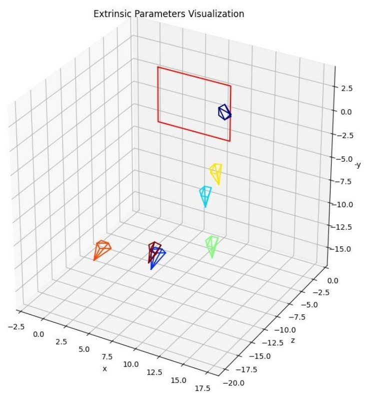
extrinsics

```
 [[-0.48487718  0.07918298  0.30690069 -6.25161791 21.16255213
45.65064338]
 [-0.63959615  0.18640668  1.14109074 -3.99491613 21.16204995
48.16348544]
 [-1.22515302  0.07078702  0.3314717  -6.65225027 16.42002213 39.29925478]
 [-1.10895678  0.63174322  1.06973368 -3.4841274  20.14753331 43.79878306]
 [-0.92067127  1.57323744  1.44970367  1.64915894  9.12013959 17.39104771]
 [-1.34068371  0.80950518  0.91891495 -1.15642457  9.20924927 22.58720479]
 [-1.14080539  0.56831172  1.13138931 -0.10079227 10.79855406
28.46234837]]
```

Visualization

opencv v.s. *our work*





Discussion

The result datum indicate that there are still some errors between the output of OpenCV and that of our implementation.

Despite the errors, our implementation is close to the OpenCV's answer, as the visualization showing that the images output are similar, which the camera's positions are likely.

The experiment result data shows that the intrinsic matrix and the extrinsic matrix have errors. This is caused by errors from linear algebra computing, as well as noise of photos. During calculating homography matrix **H** and matrix **B** ($K^{-T}K^{-1}$, as intrinsic matrix K), our implementation solve the answer by via getting the right singular vector of the matrix associated with the smallest singular value, which only uses constants, pixel numbers, object positions, resulting in the poorly conditioned numerical matrix. Much better result to solve the matrix **H** and **B** can be achieved by performing data normalization before computing the solution of **H** and **B**.

Moreover, to refine the homography matrix **H**, the method, *Nonlinear optimizer Levenberg Marquardt,* can be used. It uses minimizer function to minimize some function and the jacobian makes the optimization process more efficient. By using the method, It would get better result of homography matrix **H.**

In our implementation, the radial lens distortion is also not modeled. In contrast, OpenCV does handle the distortion.

In addition, it is inevitable that there are noise by the photos, so that it is hard to find the perfect intrinsic and extrinsic matrix. OpenCV has more optimizations to handle many case of noise, which is also a point that it does better than our implementation.

Conclusion

The basic computer vision task is to extract 3D space information from given 2D images. In order to reconstruct or recognize objects, it is essential to know the relationship between 3D coordinates and their corresponding image coordinates. Camera calibration is the fundamental of those techniques.

During the implementation, we refer the origin paper, “A flexible new technique for camera calibration,” authored by Zhengyou Zhang [1] . The paper proposed many methods but we only implement the basic ones. We strongly confirm that if we finish all of them, we can get a better result. Meanwhile, there are more new techniques presented other than this paper, which make the calibration processing more efficient and more correct.

Reference

1. Zhengyou Zhang. 2000. A Flexible New Technique for Camera Calibration. IEEE Trans. Pattern Anal. Mach. Intell. 22, 11 (November 2000), 1330–1334.

Work Plan

1. All: each one implement his own code
2. An-Chi: experimental result, discussion, conclusion
3. RUO-LIN: implementation procedure, conclusion
4. Shao-Yun: Introduction, discussion